Intelligent Autonomous Systems Project 3: SLAM

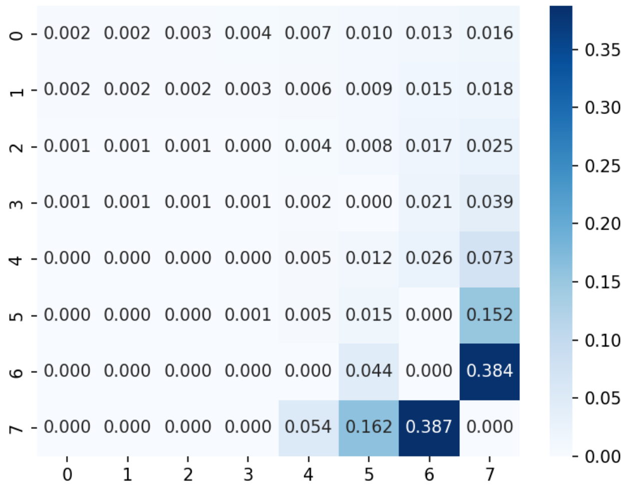
Ethan Glaser

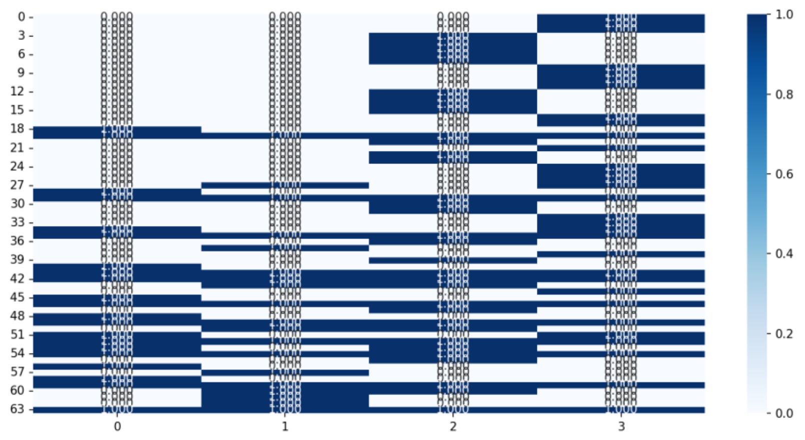
eg492

I. Problem 1

The Value/Policy Iteration component of this project involved a policy iteration approach consisting of a policy evaluation step and a q value update. The implementation was very similar to what was described in the in-class demonstration. The policy evaluation step involved iterating through each state and updating the value associated with that state based on the potential actions that were possible from that state as well as associated consequences. Once the value function was updated, the q values were changed accordingly. This back-and-forth iteration continued until approximate convergence, and then an argmax was applied to determine a policy for each state. The q values are included in the submitted zip file (q_values1.npy).

This algorithm was applied towards the 8x8 frozen lake with slipping, and the determined policy is included in the submitted zip file (policy1.npy). The optimal policy aims to keep the person out of situations where they are in a location in the same row and column as a hole, as this could lead to negative reward if slipping occurs. Simulation yields that the policy keeps movement in the top 2 rows and far right column, essentially hugging the wall the entire time until the person finally slips their way down the column to the destination. By doing so, they avoid any chance of moving leftward towards potential danger.





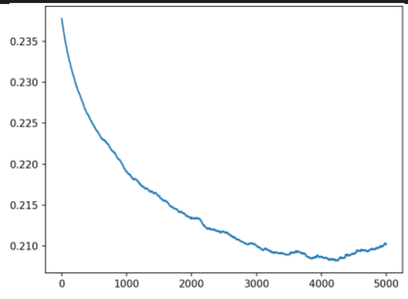
Value and policy learned from Problem 1

II. Problem2

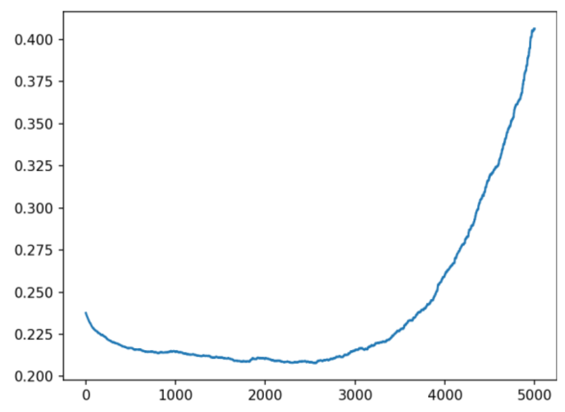
Q Learning involves a balance of exploration and exploitation that shifts throughout the training process using a parameter, epsilon. At the beginning when knowledge of the space is limited, training focuses on exploration – randomly selecting actions and updating correspondingly based on the results of those actions. Once more exploration occurs, then the knowledge is used during exploitation, enabling the affirmation or rejection of the learning so far. Problem 2 again involves the frozen lake space, with defined states. Learning the Q values involved updating the q table at each iteration based on the current state, with certain states being rewarded or punished. Hyperparameters are set to determine the balance of exploration and exploitation over time, the training length, evaluation constraints, etc.

During training, the main hyperparameters that were experimented upon were those associated with the learning rate and Q value updates – alpha and gamma. Their influence on the distance between the optimal q values determined in part 1 is displayed below, with all other hyperparameters set. The training duration was also experimented on and lengthened a bit in cases where the learning rate was low, with epsilon decay updated accordingly. The optimal policy learned during Q learning is nearly identical to the one learned in policy iteration, and both the policy and q values are saved in the zip folder.

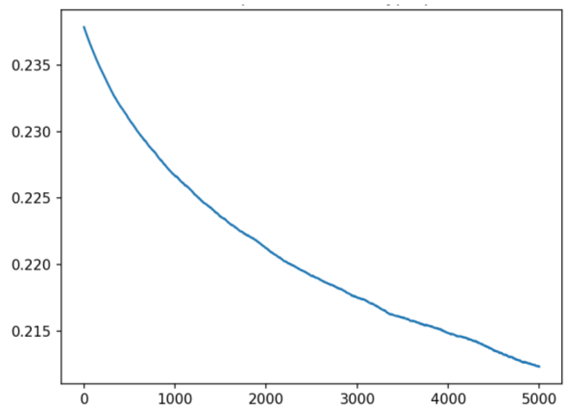
Default hyperparameters: `alpha=0.1, max_epsilon=1.0, min_epsilon=0.01, epsilon_decay_rate=0.0001, eval_itrs=50, num_training_episodes=5000, num_eval_episodes=50, gamma=1.0`



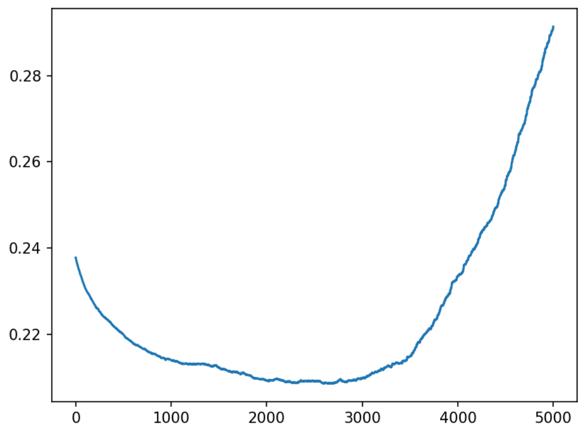
Alpha = 0.1, gamma = 1.0



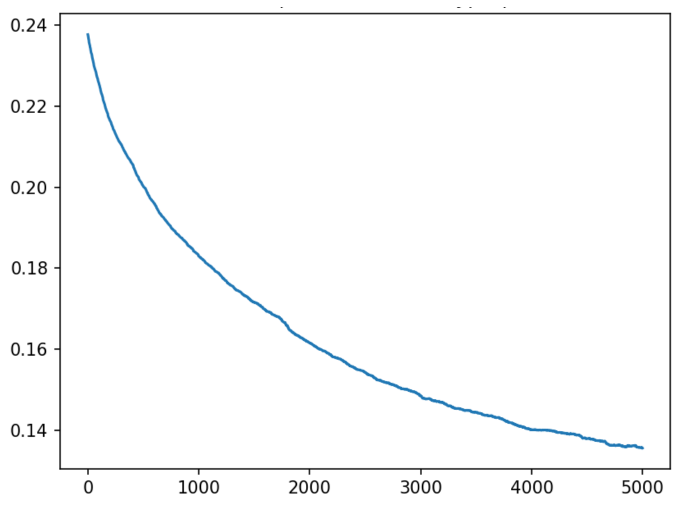
Alpha=0.1



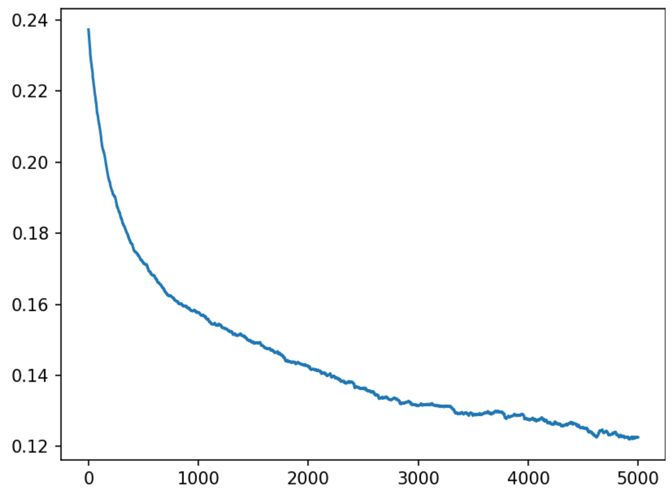
Alpha = 0.01



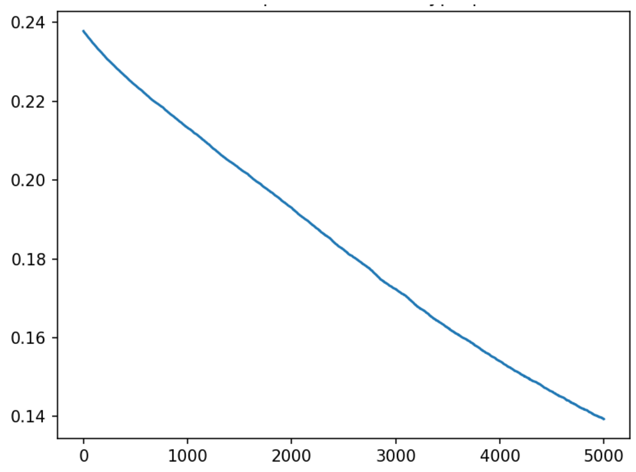
alpha = 0.05



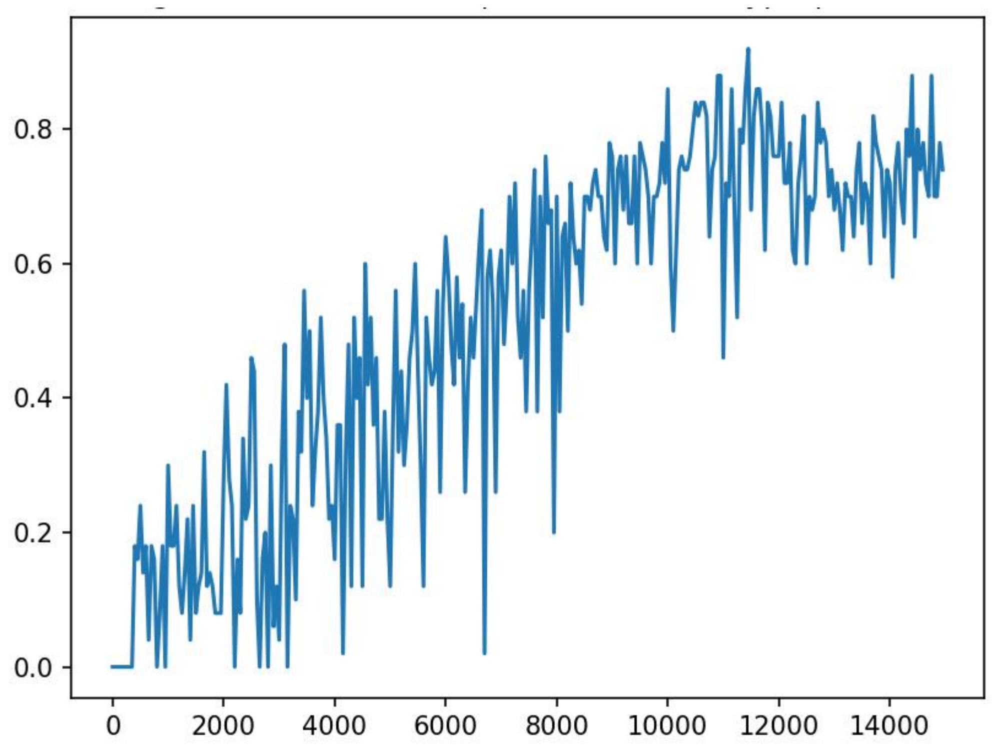
Alpha = 0.03, gamma = 0.9



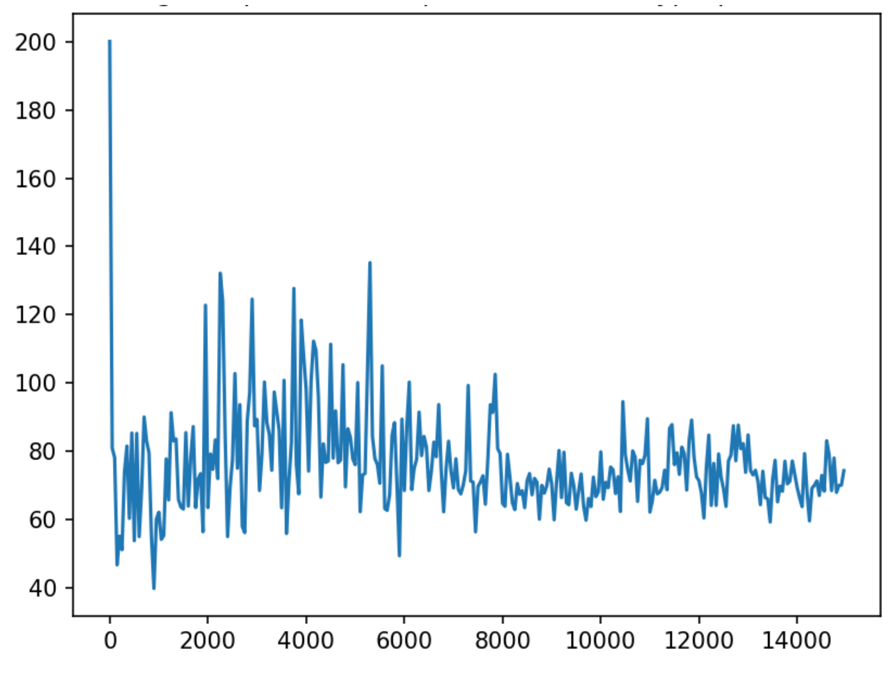
Alpha = 0.1, gamma = 0.9



Alpha = 0.01, gamma = 0.95



Average Reward across 50 evaluations during 5000 training epochs (alpha=0.01, gamma=0.95, iters=15k)



Average steps across 50 evaluations during 5000 training epochs

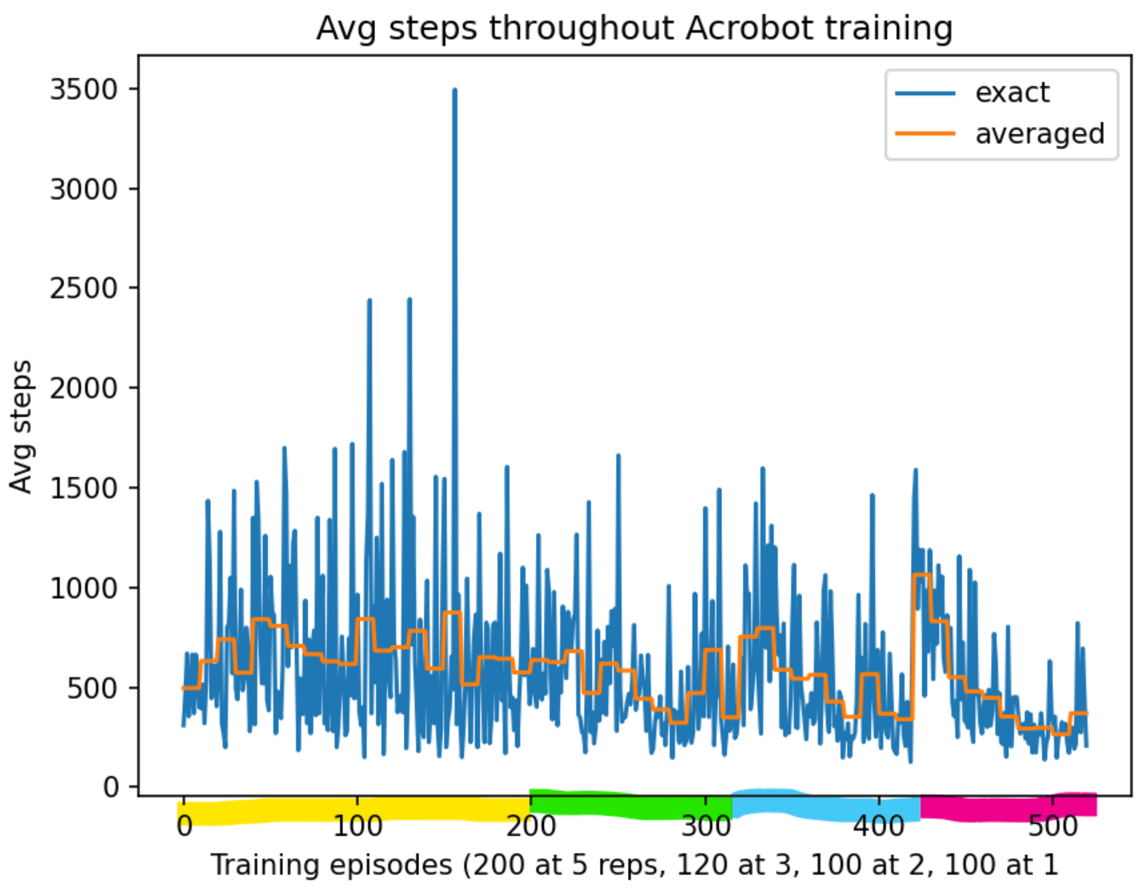
III. Problem 3

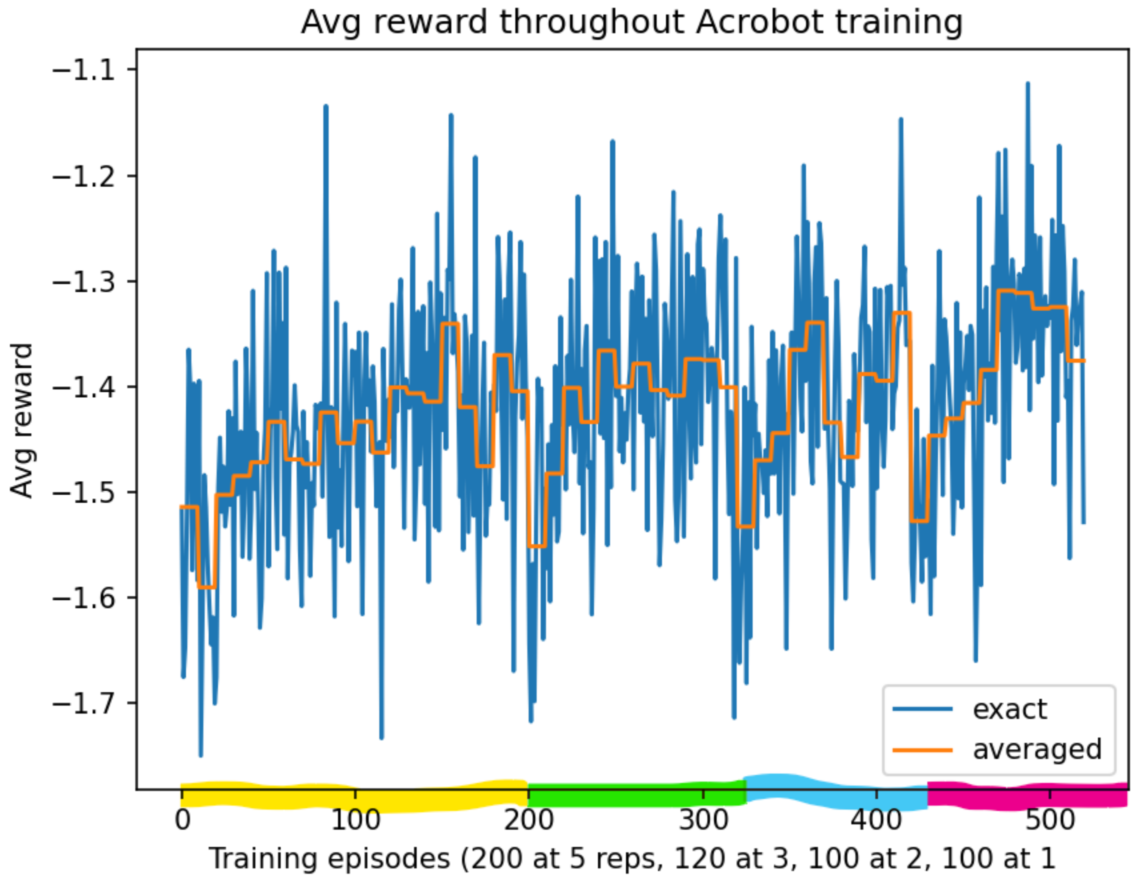
The initial approach to both the Mountain Car and Acrobot RL Q Learning problems was to set up a training scheme similar to the frozen lake after defining states using state variables (velocity, position, etc.) and putting them into bins as a way to discretize them. Once this was complete, some hyperparameters were modified but training was nowhere close to successfully happening.

Several modifications were made to the original process in order to be conducive to creating a successful policy. First, an iterative training process was conducted, with repetitions to actions included to help the early training stages converge. Many iterations were conducted on 5-repeats, then switched to 3-repeats with some knowledge, then 2-repeats with more learning, then single iterations with the knowledge learned from the repetitions so that there wasn’t a cold start and convergence was more feasible. This improved training times and made it possible for the single repetition case to converge. However this policy still wasn’t working very well, so the reward functions were modified for both the Acrobot (negative sum of cos(theta1) and cos(theta2)) and the mountain car (velocity squared). These reduced the sparsity of the reward, making learning possible at any state and drastically increasing the training speed. Videos of the final policies are in the submitted zip file.

If I was to do this differently I would experiment more with alpha and gamma values since this would provide some differentiation in the way the q values are updated.

Plots for the Acrobot are displayed below. I hadn’t implemented the value tracking while mountain car was training, which took several hours, but the overall trends are very similar. Note that the highlighted sections of the x-axis represent different repetition values, so improvements are made as a policy is learned with each number of repetitions, then a setback occurs when this value changes.


Avg steps throughout Acrobot training
exact
averaged
Avg steps
3500
3000
2500
2000
1500
1000
500
0
0
100
200
300
400
500
Training episodes (200 at 5 reps, 120 at 3, 100 at 2, 100 at 1


Avg reward throughout Acrobot training
Avg reward
−1.1
−1.2
−1.3
−1.4
−1.5
−1.6
−1.7
0
100
200
300
400
500
Training episodes (200 at 5 reps, 120 at 3, 100 at 2, 100 at 1
exact
averaged